ELSEVIER

Theoretical Computer Science 237 (2000) 327–345

Theoretical Computer Science

www.elsevier.com/locate/tcs

on node-named JAGs ☆

Chung Keung Poon

Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong, People's Republic of China

Communicated by O. Watanabe

Abstract

Given a directed graph G and two of its nodes s and t, the directed st-connectivity problem (STCON) is to decide whether there is a directed path from s to t in G. Establishing a lower bound for STCON on a general computation model such as Turing machine or branching program has been a big challenge in complexity theory. As an intermediate step, Cook and Rackoff (SIAM J. Comput. 9(3) (1980) 636–652) introduce a structured model called JAG (jumping automaton for graphs) and prove a space lower bound of $\Omega(\log^2 n/\log\log n)$ on this model. Berman and Simon (Proc. 24th Annual Symp. on Foundations of Computer science, IEEE press, New York, Tucson, AZ, November 1983) show a similar lower bound for a probabilistic JAG. We take a step further by introducing a stronger model called NN-JAG (Node-named JAG) and obtain the same space lower bound of $\Omega(\log^2 n/\log\log n)$. On a probabilistic NNJAG, we show that $S = \Omega(\log^2 n/(\log\log n + \log\log T))$ where S is the space and T is the expected time used, respectively. This gives the best expected time lower bound on this model when the space S is $O(\log^2 n/\log\log n)$.

Keywords: Graph; Connectivity; Lower bound; Space; Complexity

1. Introduction

Given a directed graph G and two of its nodes s and t, the directed st-connectivity problem (STCON) is to decide whether there is a directed path from s to t in G. It has been attracting lots of attention for many years because it is the natural abstraction for many search problems. Moreover, STCON is complete for the complexity class $\mathcal{NL}$

☆ This research is partially supported by Texas Advanced Research Projects Grant 003658386. Major portion of it was done while the author was at the Department of Computer Science, University of Toronto, Canada.

E-mail address: ckpoon@cs.cityu.edu.hk (C.K. Poon).

0304-3975/00/$ - see front matter
PII: S0304-3975(00)00019-0

(*non-deterministic logspace*) under logspace reduction. Determining the space complexity of this problem in a deterministic and probabilistic Turing machine is the key to answering such basic questions as whether nondeterminism helps in space-bounded computations and whether it can be replaced by a random source. These are the space analogoues of the $\mathcal{NP}$ vs. $\mathcal{P}$ and $\mathcal{NP}$ vs. $\mathcal{RP}$ questions.

1.1. Previous work

Currently, the smallest space achievable on a deterministic Turing machine is $O(\log^2 n)$ due to Savitch's algorithm [16] and no super-logarithmic space lower bound is known on this general computation model. Closing the gap between the upper and lower bounds seems difficult. On a probabilistic Turing machine, STCON can be solved with 1-sided error in $O(\log n)$ space using a repeated *random walk* strategy [10]. However, the method takes $O(n^n)$ expected time (and infinite worst case time). Under the constraint of bounded worst case time, the best-known upper and lower space bounds on a probabilistic Turing machine are still $O(\log^2 n)$ and $\Omega(\log n)$, respectively.

As an intermediate step in determining the space complexity of STCON, Cook and Rackoff [6] introduce a model called JAG (jumping automaton for graphs) which captures the structure of most reasonable algorithms for this problem. A JAG consists of a finite-state control and a set of pebbles which move around in the input graph. The machine can tell the *coincidence partition* of the pebbles, defined by two pebbles being in the same block of the partition if and only if they are on the same node. However, it cannot actually see the names of the pebbled nodes. Based on that limited information, it tells a pebble to *walk* from a node to an adjacent node or to *jump* to another pebble, so as to gather more information about the input graph.

Although it lacks some of the features available on a Turing machine, a JAG is strong enough to simulate, within the same time and space complexities, most of the representative algorithms for STCON and related problems. For example, depth-first search, breadth-first search and random walk on graphs [1] can be executed on a JAG directly. The simulations of the Savitch algorithm and the Barnes et al. algorithm [2], which look impossible at first sight, can be found in [6, 14], respectively. To the author's knowledge, all known deterministic algorithms for directed STCON can be implemented on a JAG.

Futhermore, the structure of a JAG allows one to measure, as the computation proceeds, how much progress the machine has made towards determining the *st*-connectivity of a graph. This enables researchers to prove a number of interesting lower bounds. First, Cook and Rackoff [6] prove a space lower bound of $\Omega(\log^2 n/\log\log n)$ which almost matches the upper bound of [16]. It is then extended to a probabilistic JAG model by Berman and Simon [5]. Their result implies that any probabilistic JAG that solves STCON with 2-sided error in $2^{\log^{O(1)} n}$ expected time requires $\Omega(\log^2 n/\log\log n)$ space.

1.2. Our work

In view of the above results, one may try to exploit those features not available on a JAG in order to obtain a better algorithm. Another direction is to add some of these features to a JAG and prove nontrivial (super-logarithmic) lower bounds on the resulting model. Here we take the second approach by introducing a stronger model called an NNJAG (Node-named JAG) and establishing lower bounds similar to that in [6, 5]. An NNJAG is the same as a JAG except that it can see the names of the pebbled nodes instead of just knowing the coincidence partition of the pebbles. This makes the machine more natural and handy for the design of graph algorithms. For example, it can compute the *mod* functions on the node names. As a result, the implementation of the Barnes et al. algorithm on an NNJAG is much easier than that on a JAG.

As another example of how an NNJAG can be stronger than a JAG, Barnes and Edmonds [3] show that any JAG that solves STCON with p pebbles in T steps must have $pT \in \Omega(n^2/\log^2 n)$. The result holds even if the JAG has an infinite number of states and the input is restricted to a particular class of graphs. However, an NNJAG with an infinite number of states can traverse any graph in this class in $O(n \log n)$ time even with only one pebble.

More importantly, a nondeterministic NNJAG has comparable power with a nondeterministic Turing machine, at least in terms of solving graph problems. Specifically, a nondeterministic NNJAG can simulate the Immerman/Szelepcsényi's algorithm [11, 18] for $\overline{\text{STCON}}$ (i.e., the complement problem of STCON) by making use of the node ordering information. Using similar idea, it can simulate a *strong jump* (i.e., jumping of a pebble from its current node to the next higher node according to some fixed numbering of the nodes) as long as all nodes are reachable from s [13]. With a small change in the model, it can in fact simulate strong jump on any arbitrary graph. (Our lower bounds still carry through in this enhanced model.) This in turn implies that it is no less powerful than a nondeterministic Turing machine with respect to solving graph problems [17]. Without access to the node ordering information, it is not clear how a nondeterministic JAG can do any of the above. Thus, a space lower bound on an NNJAG seems to be a stronger evidence (compared with one on a JAG) to the conjecture that nondeterminism helps in space bounded computations.

In this paper, we proved a space lower bound of $\Omega(\log^2 n/\log\log n)$ for STCON on a deterministic NNJAG. This extends the JAG lower bound of [6]. We also show that any probabilistic NNJAG that solves STCON with 2-sided error and expected time T requires space $S = \Omega(\log^2 n/(\log\log n + \log\log T))$. This extends the probabilistic JAG lower bound of [5]. Our proof is somewhat simpler than that in [6, 5, 13]. Moreover it shows that the same kind of graphs are difficult for both the deterministic and probabilistic NNJAGs. This may be an indication that a random source is not too useful to an NNJAG.

After our preliminary version in [13], Edmonds et al. [8] obtain a lower bound of $T = 2^{\Omega(\log^2(n/S))}$ on the expected time of a probabilistic NNJAG when $S = O(n^{\varepsilon})$ for any positive $\varepsilon < 1$. As a corollary, they also obtain a space lower bound of $\Omega(\log^2 n)$ on a

deterministic NNJAG. Compared with this result, ours is better when $S = O(\log^2 n/\log\log n)$. For example, we have that $S = O(\log n)$ implies $T = 2^{n^{\Omega(1)}}$ and that $T = 2^{\log^{O(1)} n}$ implies $S = \Omega(\log^2 n/\log\log n)$.

It is also interesting to compare our result with that of Etessami and Immerman [9]. They show that STCON cannot be solved in a model which is strictly more powerful than a deterministic JAG with $O(\log n)$ space. However, it is not clear whether their model is as strong as a JAG with $\omega(\log n)$ space or an NNJAG with $O(\log n)$ space.

In the following section, we formally define the (deterministic) JAG and NNJAG models. Section 3 presents a space lower bound on deterministic NNJAGs and Section 4 describes its extension to probabilistic NNJAGs. Finally, Section 5 discusses some open problems.

2. The JAG and NNJAG models

A (deterministic) JAG J is an automaton consisting of a set of p distinguishable pebbles, a set of q states and a transition function δ. The input to J is a triple (G,s,t) where G is a directed graph containing nodes s and t. We require that every input graph G for J has exactly n nodes and that n is known to J. To solve a graph problem such as STCON, we will have a sequence of JAGs, one for each value of n and these JAGs need not bear any similarity among them.[1] We also require that the nodes in G are labelled from 0 up to $n-1$ and that for each node, its out-edges are labelled by consecutive integers starting at 0.

We define the *instantaneous description* (id) of J as the pair (Q,Π) where Q is the current state and Π is a mapping of pebbles to nodes specifying the current location of each pebble in the graph. When J is in id (Q,Π), the transition function δ determines the next move for J based on (1) the state Q and (2) the coincidence partition of the pebbles according to the mapping Π. A move is either a *walk or a jump*. A walk (P,i,Q') consists of moving pebble P along the edge labelled i that comes out of the node $\Pi(P)$ and then assuming state Q'. (If there is no such edge, the pebble just remains on the same node.) A jump (P,P',Q') consists of moving pebble P to the node $\Pi(P')$ and then assuming state Q'.

One can think of a pebble being on a node as the scenario when a node name is stored on the tape of a Turing machine. A JAG being in a certain state reflects the other types of information (such as a loop counter, a stack) stored by the Turing machine. Walking a pebble allows the JAG to access the adjacency list of a node to obtain another node, and jumping a pebble corresponds to copying a node name from one part of the tape into another. The fact that the transition function δ depends on the coincidence partition of pebbles gives the JAG the ability to perform $(=,\neq)$-comparisons on the node names.

[1] In this way, we are defining the JAG model as a non-uniform model. It is not difficult to define a uniform JAG model. See [15] or [9] for example. We prefer the non-uniform one because lower bounds on non-uniform models carry over to uniform models.

We start J with state Q_0 and with pebbles $P_1,\ldots,P_{p-1}$ on node s and pebble P_p on node t (which makes node t distinguishable from the rest). It is said to *accept* an input (G,s,t) if it enters an accepting state on this input. It solves STCON for n-node graphs if for every input (G,s,t) where G is an n-node directed graph, it accepts the input if and only if there is a directed path from s to t in G. We define the space used by the JAG J as $p\log n+\log q$, i.e., the number of bits to specify an id. The time used is the number of moves it has made.

An NNJAG is the same as a JAG except that the transition function determines the next move based on (1) the state Q and (2) the mapping Π. Hence an NNJAG can compute any arbitrary function on the names of those pebbled nodes in order to decide its next move. For simplicity, we also make the following technical changes. Firstly, we assume that the labels of nodes s and t are always fixed (say, as 0 and $n-1$, respectively). Hence s and t are not part of the input. Secondly, there is no need to mark node t with a special pebble because the pebbles "can see" the label of the nodes they are sitting on. Therefore all the pebbles are on node s initially.

It should be noted that an NNJAG is still a restricted computation model because it does not have random access to the input. For example, before it can detect whether there is an edge from node u to v, it must move a pebble to u. If we allowed an NNJAG to perform strong jumps as described in the introduction, the machine would have random access to the input and would be at least as powerful as a general model like Turing machine in dealing with graph problems. Unfortunately, we are not able to prove any super-logarithmic lower bound on such a model yet.

3. Lower bound for deterministic NNJAGs

The main result in this section is a lower bound for STCON on a deterministic NNJAG. The idea is to reduce the problem of determining the *st*-connectivity of an input graph to that of traversing a path from s to t. The reduction is possible because of the structure of an NNJAG.

Theorem 3.1. *Any NNJAG that solves* STCON *for every n-node graph requires at least* $\log^2 n/11\log\log n$ *space.*

Proof. Let J be an NNJAG that solves STCON for every n-node graph with p pebbles and q states. Since an NNJAG with only one state is not very interesting, we assume that $pq\geqslant 2$. From Lemma 3.2, to be proved below, there is a tree G with $(38p^2\log(pq)+1)^p$ nodes such that there is a path from node s to node t; and when G is fed as input to J, node t is never visited by any pebble. That means none of the in-edges of t has been traversed. Let G' be the graph obtained by removing all the in-edges of t in G. Then t is an isolated node in G'. However, J cannot tell the difference between G and G' as none of the in-edges of t in G has been tried. Hence it must give the wrong answer for either G or G'. It follows that any

NNJAG J solving STCON for n-node graphs with p pebbles and q states must satisfy $n<(38p^2\log(pq)+1)^p$ or $p>\log n/\log(38p^2\log(pq)+1)$. If both $p<\log n$ and $\log q<\log^2 n$, we can show that $p>\log n/11\log\log n$. Therefore the space required by J is $p\log n+\log q>\log^2 n/11\log\log n$. □

Lemma 3.2. *For every NNJAG J with p pebbles and q states such that $pq\geqslant 2$, there is a tree G with at most $(38p^2\log(pq)+1)^p$ nodes and out-degree at most two such that G is st-connected but if J starts with all pebbles on node s, no pebble will ever visit node t during the computation.*

Following the proof technique of [6, 5, 13], we prove Lemma 3.2 by first defining a class of graphs called *skinny trees* from which the worst case graph for the NNJAG will be chosen. Our graphs are recursively constructed as in [6, 5] and are simpler than those in [5]. Then we characterize a subcomputation of an NNJAG on this type of graph according to the level of "cooperation" among the pebbles during this subcomputation. Finally, we prove an inductive statement on subcomputations of various levels of pebble cooperation from which Lemma 3.2 follows trivially.

Our proof differs from [6, 5] mostly in the way we find (and view) the difficult graph. Both [6, 5] examine the behaviour of the JAG on an infinite binary tree whose nodes are without names. Then the tree is pruned to the right size carefully so as not to disturb the behaviour of the JAG. The pruning is possible because the pebbles do not know much about their positions in the tree.

On the other hand, the pebbles of an NNJAG can possibly learn a lot about their positions by the node names they saw. Our approach (or the way to view it) is to give away complete information about the positions by labelling the nodes "nicely" but make use of the large possibilities of edge connections to defeat the NNJAG. Finally, we remark that the proof here uses a simpler counting argument than that used in the preliminary version of this paper [13].

3.1. The skinny trees

Let d be an integer parameter to be chosen later and let $\alpha\in\{0,1\}^d$. Denote by $\alpha[i]$ the ith bit of α. We take $G(\alpha)$ as a binary tree with vertex set $V=\{v_s\}\cup\{v_{\langle 1,0\rangle},\ldots,v_{\langle d,0\rangle}\}\cup\{v_{\langle 1,1\rangle},\ldots,v_{\langle d,1\rangle}\}$ and edge set $E=E_0\cup E_1$ where $E_0=\{(v_s,v_{\langle 1,0\rangle})\}\cup\{(v_{\langle i,\alpha[i]\rangle},v_{\langle i+1,0\rangle})\mid 1\leqslant i<d\}$ and $E_1=\{(v_s,v_{\langle 1,1\rangle})\}\cup\{(v_{\langle i,\alpha[i]\rangle},v_{\langle i+1,1\rangle})\mid 1\leqslant i<d\}$. Edges in E_0 have label 0 and edges in E_1 have label 1. (Labelling of nodes will be dealt with later.) The node v_s and $v_{\langle d,\alpha[d]\rangle}$ are called the *start* and *goal* of $G(\alpha)$, respectively. Thus $\alpha[1],\ldots,\alpha[d-1]$ determine the edge connections in $G(\alpha)$ while $\alpha[d]$ determines which of the nodes $v_{\langle d,0\rangle},v_{\langle d,1\rangle}$ will be the goal of $G(\alpha)$. See Fig. 1 for an example of $G(\alpha)$.

$G(\alpha_1,\ldots,\alpha_k)$ is obtained by replacing each node u in the graph $G(\alpha_k)$ with a copy of $G(\alpha_1,\ldots,\alpha_{k-1})$ which will be referred to as $G^u(\alpha_1,\ldots,\alpha_{k-1})$. Furthermore, each edge (u,v) in $G(\alpha_k)$ is replaced by the edge (u',v') where $u'=$ goal of $G^u(\alpha_1,\ldots,\alpha_{k-1})$ and

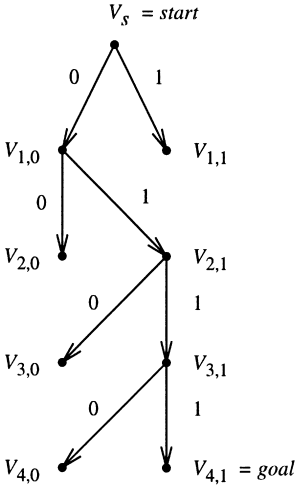

Fig. 1. $G(\alpha)$ with $d=4$ and $\alpha=0111$.

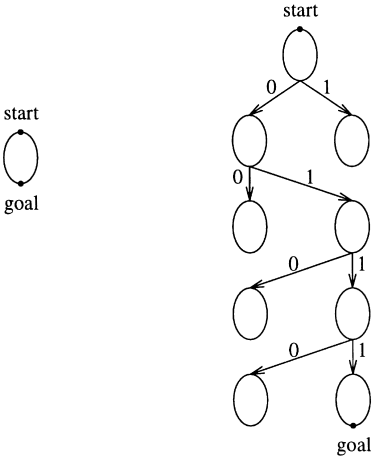

Fig. 2. Left: a copy of $G(\alpha_1,\ldots,\alpha_{k-1})$. Right: $G(\alpha_1,\ldots,\alpha_k)$ with $d=4$ and $\alpha_k=0111$.

$v'=$ start of $G^v(\alpha_1,\ldots,\alpha_{k-1})$. The label on (u',v') is the same as that of (u,v). The start of $G^{start}(\alpha_1,\ldots,\alpha_{k-1})$ and the goal of $G^{goal}(\alpha_1,\ldots,\alpha_{k-1})$ are chosen as the *start* and *goal* of $G(\alpha_1,\ldots,\alpha_{k-1})$ respectively. See Fig. 2 for an example of $G(\alpha_1,\ldots,\alpha_k)$.

The family of *skinny* trees is the set of graphs $\{G(\alpha_1,\ldots,\alpha_p)\,|\,\alpha_1,\ldots,\alpha_p\in\{0,1\}^d\}$. Each graph in this family has $(2d+1)^p$ nodes which are labelled from 0 up to $(2d+1)^p-1$ in a way to confuse the NNJAG. Suppose we label the nodes of a tree by the depth-first ordering. Then the tree can be searched in $O(p\log(2d+1))$ space. On the contrary, labeling the tree in the breadth-first ordering does not seem to help the searching. In other words, telling the NNJAG the depth of each pebble in the tree (i.e., its distance from the start node) does not seem to be useful.

Thus, we shall label the graphs so that for every node label l and every edge label j,

L1 the distance of the node with label l from the start is the same in every skinny tree, and
L2 the label on the destination node of the edge with label j from the node with label l in G is fixed for every skinny tree G (provided such an edge exists in G).

To achieve that, we simply label a node u by a p-digit number of base $2d+1$ according to where it is in $G(\alpha_1,\ldots,\alpha_p)$ for each level of recursion. More precisely, for every $\alpha_1,\ldots,\alpha_p$, the label of node u in $G(\alpha_1,\ldots,\alpha_p)$ is $n_1n_2\cdots n_p$, where each digit n_i represents a vertex $v_i \in \{v_s, v_{\langle 1,0\rangle},\ldots,v_{\langle d,0\rangle}, v_{\langle 1,1\rangle},\ldots,v_{\langle d,1\rangle}\}$ such that u is the node v_1 in the $G^{v_2}(\alpha_1)$ of the $G^{v_3}(\alpha_1,\alpha_2)$ of the $\ldots G^{v_p}(\alpha_1,\ldots,\alpha_{p-1})$ of $G(\alpha_1,\ldots,\alpha_p)$. It is easy to verify that a family of skinny trees such labelled satisfies (L1) and (L2).

For the deterministic case, we shall set $d = 19p^2\log(pq)$ and find a difficult input (G,s,t) for J, where G is chosen from the family of skinny trees and s,t are its start and goal.

3.2. Blocks of pebbles

It is difficult for a single pebble to traverse a copy of $G(\alpha_1)$ from its start to its goal without jumping. Whenever the pebble hits a dead end, it is stuck. In contrast, two cooperating pebbles can easily traverse a $G(\alpha_1)$; and a group of $k+1$ pebbles can easily traverse a $G(\alpha_1,\ldots,\alpha_k)$ using the following strategy. At the node along the path from the start to the goal of $G(\alpha_1,\ldots,\alpha_k)$ where two copies of $G(\alpha_1,\ldots,\alpha_{k-1})$ split, leave a single pebble (which we call the *leader* of the group). The other k pebbles, called the *helpers*, choose one copy of $G(\alpha_1,\ldots,\alpha_{k-1})$ and recursively traverse it. If the goal of this copy is a dead end, they jump back to the leader and recursively traverse the other copy. When it is done, the leader jumps to the goal of the $G(\alpha_1,\ldots,\alpha_{k-1})$ just traversed. Repeating this $d+1$ times, they traverse all the $d+1$ copies of $G(\alpha_1,\ldots,\alpha_{k-1})$'s along the path from the start to the goal of $G(\alpha_1,\ldots,\alpha_k)$. Thus cooperation among pebbles can substantially increase the power of an NNJAG. Below, we shall formalize this concept of pebble cooperation.

Consider the pebble locations at time t_0. We partition the pebbles into blocks so that two pebbles are in the same block if and only if they are on the same node at time t_0. We denote by $B(v,t_0)$ the block of pebbles on node v at time t_0.

As computation goes on, some pebbles may move apart and some may come together. For example, the k helpers in the above algorithm "walk" away from their leader to traverse a copy of $G(\alpha_1,\ldots,\alpha_{k-1})$. During the traversal, they may further split apart and jump to each other. However, they are considered helping their leader. On the other hand, if they jump to pebbles of other groups, they become helpers for the other groups. The following definition tries to identify which pebble is helping which block during the computation.

The *continuation* of a block $B(v,t_0)$ at time $t \geqslant t_0$, denoted as $B(v,t_0,t)$, is defined recursively as follows. $B(v,t_0,t_0)$ is just $B(v,t_0)$ itself. For $t > t_0$, if the move from time

$t-1$ to t causes a pebble P of $B(v,t_0,t-1)$ to hit (either by a walk or jump) some pebbles in another block continuation $B(v',t_0,t-1)$, then $B(v,t_0,t)=B(v,t_0,t-1)\backslash\{P\}$ and $B(v',t_0,t)=B(v',t_0,t-1)\cup\{P\}$. All the other block continuations are unchanged. If the move does not cause any collision, then $B(v,t_0,t)=B(v,t_0,t-1)$ for every block $B(v,t_0)$. For convenience, we shall use "block" to stand for "continuation of block" when the context is clear.

We are now ready to define a measure of pebble cooperations. Consider the subcomputation from time t_0 to t_1. It is useful to imagine that each block defined at time t_0 has a leader which is standing at some key position and a number of helpers exploring the part of graph near their leader. Since helpers can be shared among different blocks, it is more useful to measure the total number of helpers than to measure the number of helpers in each individual block. Let $b(t_0,t_1)$ be the number of blocks defined at time t_0 and remained non-empty at time t_1. Define $h(t_0,t_1)$, the number of helpers during the computation from time t_0 to t_1, as $h(t_0,t_1)=p-b(t_0,t_1)$. We shall use $h(t_0,t_1)$ as a measure of the "level of cooperation" among the pebbles during the computation from time t_0 to t_1. The larger is $h(t_0,t_1)$, the greater is the cooperation. This is one of the key ingredients in the proofs of [6, 5].

Note that $h(t_0,t_1)$ is monotonic non-decreasing in t_1. Increment takes place exactly when the last pebble (which must be the leader) of the continuation of a block $B(v,t_0)$ moves to join another block continuations (and hence becoming a helper). Once the continuation of $B(v,t_0)$ becomes empty, it cannot become non-empty again.

On the other hand, $h(t_0,t_1)$ is monotonic non-increasing in t_0. To see this, consider a time t between t_0 and t_1. The continuation of a block $B(v,t_0)$ may occupy more than a node at time t. If we partition the pebbles at time t, then $B(v,t_0,t)$ will be partitioned into more than one block. In other words, the pebble partition at time t is always a refinement of that induced by the set of blocks defined at time t_0 and remained non-empty at time t. Computation subsequent to step t preserves this refinement property. Since more non-empty blocks means fewer helpers, we have $h(t_0,t_1)\geqslant h(t,t_1)$.

3.3. k-computations

We now consider subcomputations on a skinny tree. A block $B(u,t_0)$ is said to traverse a copy of $G(\alpha_1,\ldots,\alpha_k)$ (named $G^v(\alpha_1,\ldots,\alpha_k)$, say) if it lies on or above the start of $G^v(\alpha_1,\ldots,\alpha_k)$ at time t_0 and at a later time t_1, some pebble of $B(u,t_0,t_1)$ is on the goal of $G^v(\alpha_1,\ldots,\alpha_k)$. A computation on $G(\alpha_1,\ldots,\alpha_p)$ between time t_0 and t_1 is called a k-computation, $1\leqslant k\leqslant p$, if $h(t_0,t_1)<k$. The aim of a k-computation is to traverse a copy of $G(\alpha_1,\ldots,\alpha_k)$ while having less than k helpers. Thus it is said to be *successful* if some block $B(u,t_0)$ has traversed a copy of $G(\alpha_1,\ldots,\alpha_k)$ by time t_1.

The bound on the number of helpers limits the cooperation among the pebbles. When $k=1$, the cooperation is minimal. Each block contains only one pebble (the leader) and its task is to traverse a $G(\alpha_1)$ without any helper. No pebble jumps are allowed because that will create helpers. For $k>1$, a block can have up to $k-1$ helpers but it has the harder task of traversing a $G(\alpha_1,\ldots,\alpha_k)$. At the extreme when $k=p$, the

NNJAG works with its full power because there is always at least one non-empty block (and at most $p-1$ helpers) in any computation. In fact, the whole computation of the NNJAG which starts at the fixed initial state Q_0 and with all pebbles on the start of $G(\alpha_1,\ldots,\alpha_p)$ is a p-computation. The task of that single block on the start node (i.e., node s) is to traverse the whole graph which is equivalent to sending some pebble to the goal node (i.e., node t). If we can find a setting of $\alpha_1,\ldots,\alpha_p$ such that any p-computation fails, then Lemma 3.2 is proved. The following lemma shows that we can set the α_i's incrementally so that for any $k\in[1..p]$, any k-computation fails.

Lemma 3.3. *For all $k\in[1..p]$, there exists $\alpha_1,\ldots,\alpha_k\in\{0,1\}^d$ such that for any $\alpha_{k+1},\ldots,\alpha_p\in\{0,1\}^d$, any k-computation by J on $G(\alpha_1,\ldots,\alpha_p)$ is not successful.*

We first prove Lemma 3.2 using Lemma 3.3. Since there is always at least one block in any computation, the whole computation performed by the NNJAG on $G(\alpha_1,\ldots,\alpha_p)$ is a p-computation. It follows from Lemma 3.3 that we can set $\alpha_1,\ldots,\alpha_p$ so that this p-computation is not successful. Hence the goal of $G(\alpha_1,\ldots,\alpha_p)$ is never traversed. Finally, note that the graph $G(\alpha_1,\ldots,\alpha_p)$ has size $(2d+1)^p=(38p^2\log(pq)+1)^p$. Hence Lemma 3.2 follows.

We now prove Lemma 3.3 by induction on k. Let $H(k)$ denote the statement of Lemma 3.3 for a fixed value of $k\in[1..p]$.

3.3.1. Base case: $H(1)$ is true

Recall that no pebble jump is allowed in a 1-computation (except possibly in the last step of an unsuccessful 1-computation). Therefore each pebble traces a simple path of a certain length. The following lemma shows that not many different paths can be traced in 1-computations because there are not too many different 1-computations.

Lemma 3.4. *The sequence of id's a 1-computation on $G(\alpha_1,\ldots,\alpha_p)$ goes through can be determined (independent of $\alpha_1,\ldots,\alpha_p$) by*
(A1) *the initial id (Q,Π) of the 1-computation, and*
(A2) *for every pebble P, the length of the path traced by P.*

Proof. Suppose the 1-computation starts at time t_0. We shall prove by induction on $t\geqslant t_0$ that the id at time t is uniquely determined.

(*Base case*). The id at time t_0 is (Q,Π) as specified by (A1).

(*Induction step*). Suppose that the id at time t is known to be (Q_t,Π_t) and that the 1-computation has not finished at time t. (If it has already finished, then we need not determine the id at time $t+1$ at all.) Then (Q_t,Π_t) determines the next move and hence the next state Q_{t+1}. If the move is a jump, then the new pebble locations Π_{t+1} are also fixed. Therefore, let us assume the move is a walk (P,i,Q_{t+1}). Observe that Π and Π_t tell us the depth of P at time t_0 and time t, respectively, because of the labelling property (L1). Hence their difference determines whether P has already finished the distance specified in (A2). If so, $\Pi_t(P)$ must be a leaf and hence P will remain on the same node at time $t+1$. Otherwise, $\Pi_t(P)$ must be an internal node. By

labelling property (L2), the edge label i and the node label $\Pi_t(P)$ determines which node P will go to. Therefore in both cases, Π_{t+1} can be determined. □

To determine whether a 1-computation is successful, it is sufficient to look at the initial portion in which no pebble traverses more than $2d$ edges. If a pebble traverses $2d$ edges, it must have already completed traversing a $G(\alpha_1)$. (The worst case happens when the pebble starts from a child of the start of a $G(\alpha_1)$. Then it has to traverse $d-1$ edges to reach the goal of that $G(\alpha_1)$, one edge to the start of the next $G(\alpha_1)$ and then d edges to reach the goal of that new $G(\alpha_1)$.)

We now compute an upper bound on the number of different 1-computations in which no pebble traverses more than $2d$ edges. There are $(2d+1)^p$ nodes in the graph and hence $q\times((2d+1)^p)^p = q(2d+1)^{p^2}$ choices for (A1). There are at most $(2d+1)^p$ choices for (A2). Consequently, there are at most $q(2d+1)^{p^2+p}$ different sequences of id's. (Of course, some choices of (A2) or some combinations of (A1) and (A2) may not constitute a realistic 1-computation but this only decreases the number of choices.)

In each 1-computation, at most p different paths can be traced. It follows that at most $pq(2d+1)^{p^2+p}$ α_1's (out of the total 2^d possibilities) will have a successful 1-computation for some choice of (A1) and (A2). When $d \geqslant 19p^2\log(pq)$, there exists an α_1 such that for any choice of $\alpha_2,\ldots,\alpha_p$, any 1-computation on $G(\alpha_1,\ldots,\alpha_p)$ is not successful. Hence $H(1)$ follows.

3.3.2. Induction step: $H(k) \to H(k+1)$

Throughout this subsection, we shall call each copy of $G(\alpha_1,\ldots,\alpha_k)$ a *supernode*. The structure within a supernode is determined by $\alpha_1,\ldots,\alpha_k$ and how the supernodes are connected in $G(\alpha_1,\ldots,\alpha_p)$ varies according to $\alpha_{k+1},\ldots,\alpha_p$.

Although the movement of each individual pebble in a $(k+1)$-computation can be quite complex, the movement of each block is not. First of all, each block can only traverse a tree (instead of a forest) of nodes. This can be argued by induction on t. At time t, all pebbles of $B(v,t)$ are at the node v. Suppose the statement is true at time $t' \geqslant t$, and consider a pebble P in $B(v,t,t')$ being moved to a node u outside the tree T traversed by $B(v,t)$. If P walks, then u is linked to T via the edge just traversed. If P jumps, then u must contain another pebble P' just before the jump. Moreover, P' belongs to another block continuation, otherwise u is in T. Therefore P no longer belongs to the continuation of $B(v,t)$. Hence, the statement is true at time $t'+1$. The following lemma shows an even stronger property of the tree traversed.

Lemma 3.5. *Let $\alpha_1,\ldots,\alpha_k$ be fixed such that for any $\alpha_{k+1},\ldots,\alpha_p$, any k-computation on $G(\alpha_1,\ldots,\alpha_p)$ is not successful. Let $B(v,t_0)$ be any block of a $(k+1)$-computation on $G(\alpha_1,\ldots,\alpha_p)$. Suppose the block $B(v,t_0)$ successfully completed the traversal of a supernode, $G^u(\alpha_1,\ldots,\alpha_k)$, at time t. Then, all pebbles of $B(v,t_0,t)$ lie within $G^u(\alpha_1,\ldots,\alpha_k)$.*

Proof. Consider a $(k+1)$-computation which started at time t_0 and has a block $B(v,t_0)$ which completed traversing $G^u(\alpha_1,\ldots,\alpha_k)$ at time t. Let the sequence of (actual) nodes

on the path from the start to the goal of $G^u(\alpha_1,\ldots,\alpha_k)$ be $(v_1,\ldots,v_m)$. Here v_1 and v_m denotes the start and goal of $G^u(\alpha_1,\ldots,\alpha_k)$, respectively.

Node v cannot lie below node v_1, otherwise $B(v,t_0)$ cannot traverse $G^u(\alpha_1,\ldots,\alpha_k)$. Moreover, if $v=v_1$, it is obvious that the continuation of $B(v,t_0)$ lies within $G^u(\alpha_1,\ldots,\alpha_k)$ between time t_0 and t. Hence the lemma immediately follows in this case. The whole block of $B(v,t_0)$ is within $G^u(\alpha_1,\ldots,\alpha_k)$ between time t_0 and t trivially. So let us assume that v is above v_1. Then it is possible that $B(v,t_0)$ attempts to traverse G^u as well as other supernodes simultaneously. Therefore, the key to the proof is to find out a time t_1 when $B(v,t_0)$ is "committed" to traversing $G^u(\alpha_1,\ldots,\alpha_k)$.

To be more specific, we shall exhibit a time t_1 such that

1. $B(v,t_0,t_1)$ occupies node v_1, and
2. the block $B(v_1,t_1)$ finished traversing $G^u(\alpha_1,\ldots,\alpha_k)$ at time t.

By condition (1), $B(v,t_0,t_1)$ occupies v_1 and possibly other nodes $w_1,\ldots,w_j$ which may lie inside or outside $G^u(\alpha_1,\ldots,\alpha_k)$. If we re-partition the pebbles of $B(v,t_0,t_1)$ at time t_1, we will obtain a number of blocks $B(v_1,t_1),B(w_1,t_1),\ldots,B(w_j,t_1)$. Let us call these blocks the t_1-*offsprings* of $B(v,t_0)$. We shall show by condition (2) that at time t all the offsprings except $B(v_1,t_1)$ will vanish. Since $B(v_1,t_1)$ is trivially within $G^u(\alpha_1,\ldots,\alpha_k)$ between time t_1 and t, the continuation of $B(v,t_0)$ lies within $G^u(\alpha_1,\ldots,\alpha_k)$ at time t.

To argue that the other t_1-offsprings of $B(v,t_0)$ become empty at time t, we consider the number of blocks defined at time t_0 and t_1 that remain non-empty at time t. By the choice of $\alpha_1,\ldots,\alpha_k$ and condition (2), there should be at most $p-k$ blocks defined at time t_1 that remain non-empty at time t. That is,

$$b(t_1,t)\leqslant p-k.$$

By the definition of $(k+1)$-computation, there should be at least $p-k$ blocks defined at time t_0 that remains non-empty at time t. That is,

$$p-k\leqslant b(t_0,t).$$

For each block $B(x,t_0)$ which is non-empty at time t, at least one of its t_1-offsprings must be non-empty at time t. Therefore,

$$b(t_0,t)\leqslant b(t_1,t).$$

Using these three inequalities, we conclude that $b(t_0,t)=b(t_1,t)$. Furthermore, if there is a block $B(x,t_0)$ which has more than one t_1-offspring that remains non-empty at time t, then $b(t_0,t)$ is strictly less than $b(t_1,t)$. By condition (2), $B(v_1,t_1,t)$ is non-empty. Hence all the t_1-offsprings of $B(v,t_0)$ except $B(v_1,t_1)$ is empty at time t.

To find a time t_1 that satisfies conditions (1) and (2), let P be the first pebble in the continuation of $B(v,t_0)$ that reaches the goal of $G^u(\alpha_1,\ldots,\alpha_k)$. So P is on node v_m at time t. Then we let $t_m=t$ and for $i=m-1$ down to 1, let t_i to be the smallest value such that node v_{i+1} is continuously occupied between t_i+1 and t_{i+1} inclusive. We claim that $B(v_{i+1},t_{i+1},t)\subseteq B(v_i,t_i,t)$ for $i=1,\ldots,m-1$. This is true because the only

edge going into v_{i+1} is from v_i. By the choice of t_i, the move at time t_i must be to walk some pebble from node v_i to v_{i+1}. Hence node v_{i+1} is occupied by the continuations of $B(v_i,t_i)$ at any time between t_i+1 and t_{i+1} inclusive. As $B(v_{i+1},t_{i+1})$ occupies only node v_{i+1}, $B(v_{i+1},t_{i+1}) \subseteq B(v_i,t_i,t_{i+1})$. This implies that $B(v_{i+1},t_{i+1},t) \subseteq B(v_i,t_i,t)$ and the claim follows. By the claim and the observation that $P \in B(v_m,t_m,t)$, we conclude that $P \in B(v_1,t_1,t)$. That means $B(v_1,t_1)$ completed traversing $G^u(\alpha_1,\ldots,\alpha_k)$ at time t, i.e., condition (2) is satisfied. Now observe that P is in both $B(v_1,t_1,t)$ and $B(v,t_0,t)$, i.e., $B(v_1,t_1,t)$ and $B(v,t_0,t)$ are not disjoint. This can happen only if $B(v,t_0,t_1)$ occupies node v_1. Hence condition (1) is satisfied.

A corollary of Lemma 3.5 is that each block can only traverse a "path" of supernodes (as opposed to a "tree" of supernodes) during a $(k+1)$-computation. Given the above lemma, we may say that, when looking at the whole of the $(k+1)$-computation, it tries to traverse at most $p-k$ paths of supernodes. If it has tried to traverse two or more paths, the pebbles (particularly, the k helpers) in the $(k+1)$-computation may work as follows. First they try to help a leader and try to lead the leader to the goal of a supernode. After completing their job successfully, they either may continue to help the leader or may jump to another block for helping their new leader. When jumping, they again try to lead their new leader to the goal of a supernode.

The whole work of the $(k+1)$-computation may in fact be more complicated. For example, the k helpers may very often make jumps and may try to help two or more leaders simultaneously. However, we here would like to emphasize this: it never happens that a single block can traverse successfully two or more paths (and hence a subtree) of supernodes even if it used the full power of the k helpers. To see this, suppose block $B(v,t)$ is trying to traverse both $G^u(\alpha_1,\ldots,\alpha_k)$ and $G^v(\alpha_1,\ldots,\alpha_k)$ which are the children of some supernode $G^w(\alpha_1,\ldots,\alpha_k)$. If $B(v,t)$ has completed traversing $G^u(\alpha_1,\ldots,\alpha_k)$ first, then all its pebbles are in $G^u(\alpha_1,\ldots,\alpha_k)$ and it is now impossible for $B(v,t)$ to traverse $G^v(\alpha_1,\ldots,\alpha_k)$. (Note that some nodes, but not the goal, within $G^v(\alpha_1,\ldots,\alpha_k)$ might have been touched by pebbles of $B(v,t)$ while it is traversing $G^u(\alpha_1,\ldots,\alpha_k)$.) Now we can analyse the paths of supernodes traced by blocks in a $(k+1)$-computation very much like the way we analysed the paths (of nodes) traced by pebbles in a 1-computation.

Lemma 3.6. *Let $\alpha_1,\ldots,\alpha_k$ be fixed such that for any $\alpha_{k+1},\ldots,\alpha_p$, any k-computation on $G(\alpha_1,\ldots,\alpha_p)$ is not successful. Then the sequence of id's a $(k+1)$-computation on $G(\alpha_1,\ldots,\alpha_p)$ goes through can be specified (independent of $\alpha_{k+1},\ldots,\alpha_p$) by*

(B1) *the initial id (Q,Π) of the $(k+1)$-computation, and*

(B2) *for every block $B(v,t_0)$ defined at time t_0, the number of $G(\alpha_1,\ldots,\alpha_k)$'s that $B(v,t_0)$ has successfully traversed and whether the goal of the last one is a leaf.*

Proof. Suppose the $(k+1)$-computation starts at time t_0. We can verify by induction on $t \geqslant t_0$ that the id at time t and the members in each block continuations $B(v,t_0,t)$ can be determined.

(Base case) The id at time t_0 is (Q, Π) and the blocks are defined by Π.

(Induction step) Suppose the id at time t is known to be (Q_t, Π_t) and we also know the members of each block continuations $B(v, t_0, t)$. Suppose the $(k+1)$-computation has not finished at time t and the next move is a walk (P, i, Q_{t+1}). Then again the key is to determine if $\Pi_t(P)$ is a leaf. Once this is known, the edge label i and the node label $\Pi_t(P)$ determines which node P will arrive at because of the labelling property (L2). Then we can update the members of the blocks involved.

By looking at Π_t and $\alpha_1, \ldots, \alpha_k$ (which are fixed), we know whether $\Pi_t(P)$ is the goal of a $G(\alpha_1, \ldots, \alpha_k)$ (called $G^u(\alpha_1, \ldots, \alpha_k)$ say). (Case 1) $\Pi_t(P)$ is not the goal of a $G(\alpha_1, \ldots, \alpha_k)$. Then $\alpha_1, \ldots, \alpha_k$ determines if $\Pi_t(P)$ is a leaf. (Case 2) $\Pi_t(P)$ is the goal of a $G(\alpha_1, \ldots, \alpha_k)$. Let P belong to the block $B(v, t_0, t)$. If $B(v, t_0)$ has not traversed a supernode, then it must have started inside (i.e., not on the start of) $G^u(\alpha_1, \ldots, \alpha_k)$ implying that every pebble of $B(v, t_0, t)$ are still within $G^u(\alpha_1, \ldots, \alpha_k)$. If $B(v, t_0, t)$ has traversed a supernode, again every pebble of $B(v, t_0, t)$ must lie within $G^u(\alpha_1, \ldots, \alpha_k)$ because of Lemma 3.5. If the goal of $G^u(\alpha_1, \ldots, \alpha_k)$ (i.e., $\Pi_t(P)$) is a leaf, block B cannot traverse any more $G(\alpha_1, \ldots, \alpha_k)$'s. Therefore, if the number of supernodes traversed by B so far is strictly less than that specified in (B2), then $\Pi_t(P)$ must not be a leaf. (To know how many supernodes block $B(v, t_0)$ has traversed, we look at $\Pi, \Pi_1, \ldots, \Pi_t$ and $B(v, t_0), B(v, t_0, t_0+1), \ldots, B(v, t_0, t)$ to infer the greatest depth a block has explored using the labelling property (L1).) On the other hand, if the number of supernodes traversed by $B(v, t_0)$ is equal to that in (B2), then block $B(v, t_0)$ is on the last copy of $G(\alpha_1, \ldots, \alpha_k)$ and P is on its goal. Hence (B2) determines whether $\Pi_t(P)$ is a leaf. □

Using the above lemma, we now compute an upper bound on the number of different $(k+1)$-computations in which no blocks traverses more than $2d+1$ supernodes. (A block that traverses $2d+1$ supernodes must have completed traversing a $G(\alpha_1, \ldots, \alpha_{k+1})$.) There are $q(2d+1)^{p^2}$ choices for (B1). To bound the number of choices for (B2), note that for each block there are $2d+2$ choices for the number of supernodes traversed. For each choice, there are two possible cases depending on whether the goal of the last supernode traversed is a leaf. Finally, observe that the number of blocks is at most p. Therefore, there are $(4d+4)^p$ choices for (B2). Hence by Lemma 3.6, there are at most $q(2d+1)^{p^2}(4d+4)^p$ different sequences of id's when $\alpha_1, \ldots, \alpha_k$ are fixed as in $H(k)$. To traverse a $G(\alpha_1, \ldots, \alpha_{k+1})$, a block has to traverse a path of $d+1$ supernodes which lie within that copy of $G(\alpha_1, \ldots, \alpha_{k+1})$ and the sequence of labels on the edges connecting these supernodes has to match α_{k+1}. Hence for $\alpha_1, \ldots, \alpha_k$ fixed as in $H(k)$, at most $pq(2d+1)^{p^2}(4d+4)^p \alpha_{k+1}$'s can have a successful $(k+1)$-computation for some choice of (B1) and (B2). When $d \geqslant 19p^2 \log(pq)$, there exists a choice of $\alpha_1, \ldots, \alpha_{k+1}$ such that for any $\alpha_{k+2}, \ldots, \alpha_p$, any $(k+1)$-computation on $G(\alpha_1, \ldots, \alpha_p)$ is not successful. Hence $H(k+1)$ follows. This also completes the proof of Lemma 3.3.

4. Extension to probabilistic NNJAGs

We introduce randomness to an NNJAG J by allowing the machine to access a fresh random bit each step. The transition function δ is extended so that it depends on the random bit as well. The random bit R_t obtained in step t will not be available for the future unless it is saved by the machine. Also, it cannot be available before step t. This way of adding randomness to an NNJAG is essentially the same as what [5] do for a JAG.

A probabilistic NNJAG J is said to solve STCON with 1-sided error if for every input (G,s,t), the probability (over R) of J entering an accepting state is at least $\frac{1}{2}$ when there is a path from node s to t, and 0 otherwise. Similarly, it solves STCON with 2-sided error if for every input (G,s,t), the probability (over R) of J entering an accepting state is at least $\frac{2}{3}$ when there is a path from node s to t, and at most $\frac{1}{3}$ otherwise.

To prove a lower bound for STCON on a probabilistic NNJAG, we again reduce STCON to that of traversing a path from s to t with high probability, capitalizing on the structure of an NNJAG.

Theorem 4.1. *Any probabilistic NNJAG that solves* STCON *with* 2*-sided error for* n*-node graphs using* S *space and* T *expected time satisfies* $S \geqslant \log^2 n/(11 \log \log n + \log \log T)$.

Proof. Let J be a probabilistic NNJAG that solves STCON for every n-node graph with 2-sided error using p pebbles and q states. We assume that $pq \geqslant 2$. From Lemma 4.2, to be proved below, there is a tree G with at most $(54p^2 \log(pqT)+1)^p$ nodes and out-degree at most two such that G is st-connected but if J reaches node t with probability at least $\frac{1}{4}$, its expected time on G is more than T. Thus, if J runs in expected time T, the probability that node t is visited is less than $\frac{1}{4}$. Since G is st-connected, J has to accept with probability at least $\frac{2}{3}$. Hence $Pr[J$ has not pebbled t but accepts $G] \geqslant \frac{2}{3} - \frac{1}{4} > \frac{1}{3}$. Now construct G' from G by removing all the in-edges of node t. With the same probability (i.e. $> \frac{1}{3}$), J has not pebbled node t (and hence cannot tell the difference between G and G') and will accept G'. It follows that $n < (54p^2 \log(pqT)+1)^p$. From that, we can show that $S \geqslant p \log n \geqslant \log^2 n/(11 \log \log n + \log \log T)$. □

Note that for any constant $c \geqslant 0$, $T \leqslant 2^{\log^c n}$ implies $S \geqslant \log^2 n/(11 + c) \log \log n$; and for any constant $c \geqslant 1$, $S \leqslant c \log n$ implies $T \geqslant 2^{n^{1/c - o(1)}}$.

Lemma 4.2. *For every integer* $T \geqslant 1$ *and every probabilistic NNJAG* J *with* p *pebbles and* q *states satisfying* $pq \geqslant 2$, *there is a tree* G *with at most* $(54p^2 \log(pqT) + 1)^p$ *nodes and out-degree at most two such that* G *is st-connected but if* J *starts with all pebbles on node* s *and visits node* t *with probability at least* $\frac{1}{4}$, *then the expected time of* J *on* G *is more than* T.

To prove an expected time lower bound on a probabilistic machine M, the following approach is often used. View M as a distribution $\mathscr{M}$ of deterministic machines induced

by the distribution of random strings taken by M. Then establish a lower bound on the average running time for an arbitrary (deterministic) machine from $\mathscr{M}$ on a difficult input distribution. Theorem 1 of [19] implies that this lower bound is also one for the worst case expected time of $\mathscr{M}$, the expectation being taken over $\mathscr{M}$. For completeness, we state the required corollary of that theorem below.

Lemma 4.3 (from Theorem 1 of [19]). *For any graph distribution $\mathscr{G}$ and any machine distribution $\mathscr{J}$,*

$$\min_{J\in\mathscr{J}}\{\textit{average cost of } J \textit{ on } \mathscr{G}\} \leqslant \max_{G\in\mathscr{G}}\{\textit{expected cost of } \mathscr{J} \textit{ on } G\}.$$

Unfortunately, if M is space bounded, it may be impossible to simulate it by a distribution of deterministic machines with the same space. As an example, our space lower bound for deterministic NNJAGs in the previous section implies that any deterministic NNJAG with 1 pebble and $n^{O(1)}$ states takes infinite time to discover node t for some n-node graph. Since (for a fixed value of n) there are finitely many such deterministic NNJAGs, any distribution on them has infinite expected time for the worst case input. On the other hand, a probabilistic NNJAG with the same space can discover node t with 1-sided error for any input using only $O(n^n)$ expected time. The idea is to use a repeated random walk strategy which is a combination of the random walk technique and *probabilistic counter*. See Chapter 6 of Motwani and Raghavan [12] for an explicit algorithm and a tighter analysis.

To get around this problem, [5] work directly on the probabilistic JAG to construct a worst case input graph. In contrast, we shall use an approach similar to what we described above. The technique is generally applicable in proving lower bounds on probabilistic space bounded machines. Suppose we want to prove a lower bound on the worst case expected time of a probabilistic machine M with S space which is correct with probability λ. We simulate the first cT steps of M by a distribution of deterministic machines with $S+\log(cT)$ space (and at most cT steps). Then we prove an upper bound $(1-1/c)\lambda$ on the probability of correct computation by such a deterministic machine. Using Lemma 4.3 (with probability of error as cost), M running for at most cT steps implies that it is correct with probability at most $(1-1/c)\lambda$. Hence, to be correct with probability at least λ, its expected running time must be at least $(1/c)\times cT = T$.

Proof (*of Lemma* 4.2)**.** Let $T>1$ and J be a probabilistic NNJAG with p pebbles and q states. We denote by J_R the deterministic NNJAG induced by the random string R taken by J. We shall construct a distribution $\mathscr{J}=\{J_R \mid R\in\{0,1\}^{8T}\}$ of deterministic NNJAGs as follows. Each J_R has p pebbles. For each state Q in J, J_R will have states $\langle 1,Q\rangle, \langle 2,Q\rangle, \ldots, \langle 8T,Q\rangle$. Thus it has $8qT$ states. Suppose $\delta(R_t,Q,\Pi)=(P,P',Q')$ for some $t\in[1..8T-1]$. Then the transition function, δ_R, of J_R is defined so that $\delta_R(\langle t,Q\rangle, \Pi)=(P,P',\langle t+1,Q'\rangle)$. The case is similar when $\delta(R_t,Q,\Pi)=(P,i,Q')$. Clearly, each J_R runs for at most $8T$ steps. Moreover, its computation is identical to the first $8T$ steps of J with any random string that has prefix R.

By Lemma 4.4 below, any deterministic NNJAG with p pebbles and $8qT$ states can only reach node t with probability no more than $1/8$ for inputs randomly and uniformly chosen from the set of skinny trees with d set to $27p^2\log(pqT)$. By Lemma 4.3, there exists a skinny tree such that the probability of an NNJAG J_R chosen from $\mathscr{J}$ reaching node t of this tree is at most $\frac{1}{8}$. Consider the original probabilistic NNJAG J. If it reaches node t with probability at least $\frac{1}{4}$, then with probability atleast $\frac{1}{4}-\frac{1}{8}$, it runs for more than $8T$ steps. Hence its expected running time is more than $8T/8=T$ and Lemma 4.2 follows. □

Lemma 4.4. *Fix a random string $R\in\{0,1\}^{8T}$ and hence a deterministic NNJAG J_R. Let $d=27p^2\log(pqT)$ and let $M_k(\alpha_1,\ldots,\alpha_k)$ be the event that for some $\beta_{k+1},\ldots,\beta_p$, the computation of J_R on $G(\alpha_1,\ldots,\alpha_k,\beta_{k+1},\ldots,\beta_p)$ contains a successful k-computation. Then for $\alpha_1,\ldots,\alpha_p$ chosen uniformly at random from $\{0,1\}^{dp}$, $Pr_{\alpha_1,\ldots,\alpha_p}[M_k(\alpha_1,\ldots,\alpha_k)]<k/(8p)$.*

Proof (*By induction on k*).. Let $H(k)$ be the statement of the lemma for a fixed value of $k\in[1..p]$.

(Base case) Since J_R is deterministic, we can apply Lemma 3.4 and the argument following that lemma (with q replaced by $8qT$) and deduce that there are at most $p(8qT)(2d+1)^{p^2}(4d+4)^p\alpha_1$'s (out of the 2^d possibilities) such that for some $\beta_2,\ldots,\beta_p$, the computation of J_R on $G(\alpha_1,\beta_2,\ldots,\beta_p)$ contains a successful 1-computation. Therefore,

$$\begin{aligned}
&Pr_{\alpha_1,\ldots,\alpha_p}[M_1(\alpha_1)]\\
&\quad=Pr_{\alpha_1}[M_1(\alpha_1)]\\
&\quad\leqslant p(8qT)(2d+1)^{p^2}(4d+4)^p/2^d\\
&\quad<1/(8p),
\end{aligned}$$

when $d\geqslant 27p^2\log(pqT)$. Hence $H(1)$ is true. (Induction step) Assume $H(k)$ is true and write $Pr_{\alpha_1,\ldots,\alpha_p}[\cdot]$ as $Pr[\cdot]$. Observe that

$$\begin{aligned}
&Pr[M_{k+1}(\alpha_1,\ldots,\alpha_{k+1})]\\
&\quad\leqslant Pr[M_k(\alpha_1,\ldots,\alpha_k)]+Pr[M_{k+1}(\alpha_1,\ldots,\alpha_{k+1})|\overline{M_k(\alpha_1,\ldots,\alpha_k)}].
\end{aligned}$$

By $H(k)$, the first probability is less than $k/(8p)$. Hence we just need to show that the second probability is less than $1/(8p)$. Fix an arbitrary value of $\alpha_1,\ldots,\alpha_k$ such that $M_k(R,\alpha_1,\ldots,\alpha_k)$ does not happen. Then by Lemma 3.5, each block can only traverse a path of super-nodes during a $(k+1)$-computation. By Lemma 3.6 and the argument following that lemma, we deduce that there are at most $p(8qT)(2d+1)^{p^2}(4d+4)^p\alpha_{k+1}$'s (out of the 2^d possibilities) such that for some $\beta_{k+2},\ldots,\beta_p$, the computation by J_R on

$G(\alpha_1,\ldots,\alpha_{k+1},\beta_{k+2},\ldots,\beta_p)$ contains a successful $(k+1)$-computation. Therefore,

$$\begin{aligned}
&Pr_{\alpha_1,\ldots,\alpha_p}[M_{k+1}(\alpha_1,\ldots,\alpha_{k+1}) \mid \overline{M_k(\alpha_1,\ldots,\alpha_k)}]\\
&\leqslant Pr_{\alpha_1,\ldots,\alpha_{k+1}}[M_{k+1}(\alpha_1,\ldots,\alpha_{k+1}) \mid \overline{M_k(\alpha_1,\ldots,\alpha_k)}]\\
&\leqslant \max_{\alpha_1,\ldots,\alpha_k} Pr_{\alpha_{k+1}}[M_{k+1}(\alpha_1,\ldots,\alpha_{k+1}) \mid \overline{M_k(\alpha_1,\ldots,\alpha_k)}]\\
&\leqslant 8pqT(2d+1)^{p^2}(4d+4)^p/2^d\\
&< 1/(8p),
\end{aligned}$$

when $d \geqslant 27p^2\log(pqT)$. Hence $H(k+1)$ is true and Lemma 4.4 follows. □

5. Discussions and open problems

Improving the space lower bound to $\Omega(\log^2 n)$ on the NNJAG model has been done in [8]. The same paper also proves tight time-space lower bounds for $S \in [\log^2 n, n^\varepsilon]$ for any positive $\varepsilon < 1$. Thus, the obvious remaining open problem is to prove non-trivial space or time-space lower bounds on a more general machine model. Any such lower bounds on a Turing machine or branching program would be a breakthrough in complexity theory. An easier task is to come up with a model in between an ordinary NNJAG and an NNJAG with strong jump.

Etessami and Immerman [9] attack the problem from the approach of finite model theory. They show that *st*-connectivity cannot be expressed as a first-order logic formula augmented with deterministic transitive closure operators when the input graph only has a 1-*way local ordering*. One can view a formula in such a logic as a logspace machine which is more restricted than a logspace Turing machine but less restricted than a logspace JAG. A possible approach to define stronger models is perhaps to combine the ideas in both [9] and this paper.

Other interesting open problems include proving time-space lower bounds for USTCON on an NNJAG matching those proved on a JAG by Beame et al. [4] and Edmonds [7]. This may require very different techniques than those used for JAGs and hence may give us more insight into the problem.

Acknowledgements

I would like to thank Allan Borodin and Stephen Cook for leading me to the NNJAG model; as well as Dimitris Achlioptas, Jeff Edmonds, Hisao Tamaki and Tino Tamon for their discussions and encouragement. I would also like to thank the anonymous referees who suggested many useful improvements.

References

[1] R. Aleliunas, R.M. Karp, R.J. Lipton, L. Lovász, C. Rackoff, Random walks, universal traversal sequences, and the complexity of maze problems, in: 20th Annual Symp. on Foundations of Computer Science, IEEE Press, New York, San Juan, Puerto Rico, October 1979, pp. 218–223.

[2] G. Barnes, J.F. Buss, W.L. Ruzzo, B. Schieber, A sublinear space, polynomial time algorithm for directed $s-t$ connectivity, in: Proc., Structure in Complexity Theory, 7th Annual Conf., IEEE Press, New York, Boston, MA, June 1992, pp. 27–33.

[3] G. Barnes, J. Edmonds, Time-space lower bounds for directed *s*-*t* connectivity on JAG models, 34th Annual Symp. on Foundations of Computer Science, Palo Alto, CA, November 1993, pp. 228–237.

[4] P. Beame, A. Borodin, P. Raghavan, W.L. Ruzzo, M. Tompa, Time-space trade-offs for undirected graph connectivity, SIAM J. Comput. 28(3) (1998) 1051–1072.

[5] P. Berman, J. Simon, Lower bounds on graph threading by probabilistic machines, in: 24th Annual Symp. on Foundations of Computer Science, IEEE Press, New York, Tucson, AZ, November 1983, pp. 304–311.

[6] S.A. Cook, C.W. Rackoff, Space lower bounds for maze threadability on restricted machines, SIAM J. Comput. 9(3) (1980) 636–652.

[7] J. Edmonds, Time-space trade-offs for undirected *st*-connectivity on a JAG, Proc. 25th Annual ACM Symp. on Theory of Computing, San Diego, CA, May 1993, pp. 718–727.

[8] J. Edmonds, C.K. Poon, D. Achlioptas, Tight lower bounds for *st*-connectivity on the NNJAG model, SIAM J. Comput. 28(6) (1999) 2257–2284.

[9] K. Etessami, N. Immerman, Reachability and the power of local ordering, in: 11th Annual Symp. on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, vol. 775, Springer, Berlin, February 1994, pp. 123–135.

[10] J. Gill, Computational complexity of probabilistic Turing machines, SIAM J. Comput. 6(4) (1977) 675–695.

[11] N. Immerman, Nondeterministic space is closed under complement, Technical Report 552, Yale University, July 1987.

[12] R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge University Press, Cambridge, 1995.

[13] C.K. Poon, Space bounds for graph connectivity problems on node-named JAGs and node-ordered JAGs, 34th Annual Symp. on Foundations of Computer Science, Palo Alto, CA, November 1993, pp. 218–227.

[14] C.K. Poon, A sublinear space, polynomial time algorithm for directed *st*-connectivity on the JAG model, Manuscript (1993).

[15] C.K. Poon, On the complexity of the *ST*-Connectivity problem, Ph.D. Thesis, University of Toronto (1996).

[16] W.J. Savitch, Relationships between nondeterministic and deterministic tape complexities, J. Comput. System Sci. 4 (2) (1970) 177–192.

[17] W.J. Savitch, Maze recognizing automata and nondeterministic tape complexity, J. Comput. System Sci. 7 (4) (1973) 389–403.

[18] R. Szelepcsényi, The method of forcing for nondeterministic automata, Acta Inform. 26 (1988) 279–284.

[19] A.C. Yao, Probabilistic computations: toward a unified measure of complexity, in: 18th Annual Symp. on Foundations of Computer Science, IEEE Press, New York, Providence, RI, October 1977, pp. 222–227.